УДК 004.5:519.854.33

А. И. Овчинников, Н. В. Медведев,
А. Ю. Быков

ПРИМЕНЕНИЕ МЕТОДА ВЕКТОРА СПАДА ДЛЯ РЕШЕНИЯ ЗАДАЧИ ПОИСКА ВАРИАНТОВ ЗАЩИТЫ ОТ УГРОЗ БЕЗОПАСНОСТИ ВЫЧИСЛИТЕЛЬНОЙ СЕТИ ПРЕДПРИЯТИЯ

Рассмотрена возможность применения метода вектора спада к решению задачи выбора вариантов защиты от угроз безопасности вычислительной сети предприятия. Решаемая задача представляет собой задачу булева программирования с нелинейными ограничениями, которая не допускает построения эффективных приближенных алгоритмов в том смысле, что получение приближенного решения с заданной оценкой точности столь же сложно, как и получение точного решения. Реализация алгоритма метода выполнена на языке Java. Проведены анализ быстродействия алгоритма и оценка точности полученных решений.

Исключительно важную роль в решении задач дискретного программирования играют приближенные методы. Можно указать обстоятельства, которые вызывают необходимость разработки приближенных методов дискретного программирования.

Во-первых, из-за комбинаторного характера большинства задач дискретного программирования и большой трудоемкости их решения точные методы не дают возможности получить оптимальное решение для практических задач довольно большой размерности за приемлемое время.

Во-вторых, часто на практике для задач исследования операций применяются приближенные модели, где исходные данные определяются неточно и приближенно. Это обесценивает поиск оптимальных решений, и потому часто ограничиваются приближенными решениями, удовлетворительными с точки зрения практики. На их поиск расходуется значительно меньше машинного времени и других ресурсов ЭВМ, что особенно важно, если соответствующие задачи должны решаться в реальном времени (например, в условиях функционирования АСУ).

Особенность задачи оптимального выбора вариантов защиты от угроз безопасности вычислительной сети, как и многих других задач дискретной оптимизации, то, что она является NP-полной задачей [1, 2], т.е. невозможно получить точное решение за время, значение которого является полиномом от размерности задачи. Поэтому имеет смысл рассмотреть приближенные методы, которые используются в тех случаях, когда необходимо обеспечить оперативность решения задачи или когда размерность задачи велика.

NP-полные задачи с точки зрения качества приближенного решения можно разделить на три класса [3, 4].

1. Задачи, в которых эффективно (т.е. за полиномиальное время) строятся приближенные решения с любой наперед заданной точностью. Таковы, например, задачи о рюкзаке [4], некоторые задачи теории расписаний и др.

2. Задачи, в которых эффективно строятся приближенные решения с априорной оценкой точности. При этом ошибка может быть заранее оценена до работы алгоритма (в отличие от предыдущего случая ее нельзя устремить к нулю). Таковы, в частности, многие задачи о покрытии и др.

3. Задачи, вообще не допускающие эффективных приближенных алгоритмов: получить приближенное решение с заданной оценкой точности столь же сложно, как и точное решение. К этому классу принадлежат задача о коммивояжере, общая задача булева программирования, задачи о размещениях и многие другие [3].

Как показывает анализ, задача оптимального выбора вариантов защиты вычислительной сети относится к третьему классу.

Для решения данной задачи можно использовать следующие классы приближенных методов: методы локальной оптимизации, методы случайного поиска и комбинированные методы, или методы локально-стохастической оптимизации.

Методы случайного поиска для решения задачи выбора вариантов защиты от угроз безопасности ВС предприятия не очень удобны, так как трудно организовать управляемый случайный поиск из-за особенностей ограничений.

Остановимся на рассмотрении одного из методов локальной оптимизации – метода вектора спада [5]. Достоинствами этого метода являются его простота и то, что существует аналитическое выражение для вычисления верхней оценки числа его шагов. Рассмотрим метод вектора спада применительно к решению задачи выбора вариантов защиты от угроз безопасности вычислительной сети предприятия.

Постановка задачи. Введем обозначения.

1. $A = \{a_1, a_2, \ldots, a_n\}$ — множество возможных угроз безопасности; $N = \{1, 2, \ldots, n\}$ — множество индексов угроз.

2. $B = \{b_1, b_2, \ldots, b_m\}$ — множество средств защиты от угроз безопасности; $M = \{1, 2, \ldots, m\}$ — множество индексов вариантов защиты.

3. $T = [t_0, t_{\max}]$ — рассматриваемый период функционирования.

4. $p_i, \forall i \in N, p_i \in [0, 1]$ — возможность (вероятность) проявления i-й угрозы на интервале T определяется по данным статистики или с помощью экспертов.

5. $u_i, \forall i \in N$ — средний ущерб от возможного не предотвращения i-й угрозы.

6. $c_j, j \in M$ — стоимость j-го средства защиты.

7. $v_{ij}, \forall i \in N, j \in M, v_{ij} \in [0,1]$ — возможность (вероятность) предотвращения последствий i-й угрозы с помощью j-го средства защиты, определяемая по данным статистики или с помощью экспертов.

Возможны два варианта постановки задачи:

— максимизация возможного предотвращенного ущерба при ограничении на затраты;

— минимизация затрат при ограничении на возможный предотвращенный ущерб.

Для демонстрации применения алгоритма остановимся на решении задачи минимизации затрат при ограничении на возможный предотвращенный ущерб.

Введем булеву переменную $x_j \in \{0,1\}, \forall j \in M$, тогда

$x_j = 1$, если j-е средство защиты будет применяться в вычислительной сети для защиты от тех или иных угроз;

$x_j = 0$ — в противном случае, т.е. если j-е средство не применяется.

Тогда $\overrightarrow{X}$ — вектор булевых переменных $x_j, \forall j \in M$.

Введем следующий показатель стоимости вариантов защиты от угроз безопасности:

$$C(\overrightarrow{X}) = \sum_{j \in M} c_j x_j. \tag{1}$$

Значение данного показателя необходимо минимизировать при следующем ограничении:

$$\sum_{i \in N} u_i p_i \max_{j \in M}(v_{ij} x_j) \geqslant U_{zad}. \tag{2}$$

Это ограничение определяет нижнюю границу возможного предотвращенного ущерба, где U_{zad} — его заданное значение.

Итоговое выражение математической постановки задачи минимизации затрат при ограничении на возможный предотвращенный ущерб выглядит следующим образом:

$$\begin{aligned} &C(\overrightarrow{X}) = \sum_{j \in M} c_j x_j \to \min_{\overrightarrow{X} \in \Delta^{\text{доп}}}; \\ &\Delta^{\text{доп}} : \sum_{i \in N} u_i p_i \max_{j \in M}(v_{ij} x_j) \geqslant U_{zad}. \end{aligned} \tag{3}$$

Решением задачи будет нахождение всех неизвестных компонент вектора $\overrightarrow{X}$ и выбор средств защиты b_j, для которых соответствующая компонента вектора x_j равна единице.

Описание алгоритма. Методы локальной оптимизации базируются на метризации дискретного пространства [5], на понятиях окрестности точки в дискретном пространстве и локального экстремума функции по аналогии с непрерывным пространством.

Обозначим через $O_{\Delta^{\text{доп}}}(\vec{X}, r)$ — окрестность точки $\vec{X} \in \Delta^{\text{доп}}$ радиусом $r > 0$, тогда

$$O_{\Delta^{\text{доп}}}(\vec{X}, r) = \left\{ \vec{Y} \in \Delta^{\text{доп}} | \rho\left(\vec{X}, \vec{Y}\right) \leqslant r \right\}, \qquad (4)$$

где $\rho\left(\vec{X}, \vec{Y}\right)$ — расстояние между точками $\vec{X}$ и $\vec{Y}$ в дискретном метрическом пространстве. В случае булевых переменных наиболее часто используется метрика Хэмминга [6].

Для характеристики точек окрестности определим вектор, называемый вектором спада, характеризующий изменение целевой функции при переходе в различные точки окрестности:

$$\overrightarrow{\sigma^r}(\overrightarrow{X}) = \left\| \sigma\left(\vec{X}, \vec{Y}^{(1)}\right), \sigma\left(\vec{X}, \vec{Y}^{(2)}\right), \ldots, \sigma\left(\vec{X}, \vec{Y}^{(k)}\right) \right\|^{\mathrm{T}}, \qquad (5)$$

где $\sigma\left(\overrightarrow{X}, \overrightarrow{Y}^{(i)}\right) = F\left(\overrightarrow{Y}^{(i)}\right) - F\left(\overrightarrow{X}\right), i = 1, 2, \ldots, k$; k — число точек окрестности.

Идея метода вектора спада заключается в том, что на каждом шаге рассматривается окрестность некоторой точки, соответствующей допустимому решению задачи и осуществляется переход с помощью вычисления компонент вектора спада к той точке окрестности, которая имеет наибольшее (наименьшее) значение целевой функции. Процесс продолжается до получения локального максимума (минимума) задачи.

Приведем алгоритм одной из модификаций метода вектора спада для решения задачи оптимального выбора вариантов защиты от угроз безопасности вычислительной сети.

Шаг 0 (предварительный). Выбираем начальное допустимое приближение $\vec{X}^{(0)} \in \Delta^{\text{доп}}$. Например, будем использовать все допустимые средства для защиты, в этом случае $x_j = 1, \forall j \in M$ (это решение будет всегда допустимым при корректной постановке задачи, т.е. если ограничения позволяют задать хотя бы одно допустимое решение). Задаем значение радиуса $r = 1$.

Шаг k ($k \geqslant 1$).

1. Определяем для каждой точки, которая является допустимой согласно введенным ограничениям, окрестности $O_{\Delta^{\text{доп}}}\left(\vec{X}^{(k-1)}, r\right)$ компоненты вектора спада $\sigma\left(\vec{X}^{(k-1)}, \vec{Y}\right)$, где $\vec{Y} \in O_{\Delta^{\text{доп}}}\left(\vec{X}^{(k-1)}, r\right)$.

2. Определяем точку окрестности, для которой компонента вектора спада отрицательная и является минимальной (так как решается

задача минимизации) среди всех точек окрестности, далее эту точку принимаем за следующее приближение $\vec{X}^{(k)}$ и переходим к шагу $k+1$.

3. Если такой точки не существует, то работа алгоритма завершена, на шаге $k-1$ получена точка локального минимума.

В листинге 1 представлена программа на языке Java, реализующая алгоритм метода вектора спада применительно к решению задачи выбора вариантов защиты от угроз безопасности вычислительной сети предприятия.

Сравнение алгоритма метода вектора спада с точным алгоритмом решения задачи. Проведем сравнительный анализ рассмотренного алгоритма метода вектора спада с точным алгоритмом решения задачи. В качестве точного алгоритма выберем аддитивный алгоритм, разработанный Э. Балашем [7].

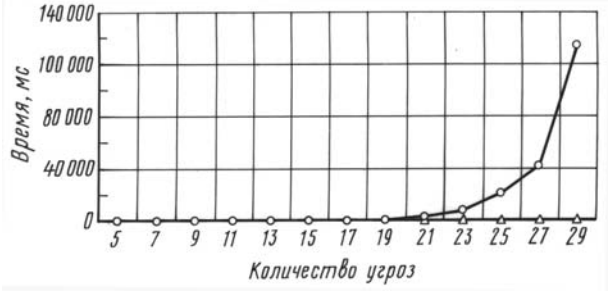
Для оценки точности полученных результатов и скорости работы алгоритма были псевдослучайным образом сгенерированы тестовые наборы данных, которые поступили на вход обоих алгоритмов. В целях усреднения результатов измерения генерировались по 5 различных наборов данных для каждого фиксированного количества средств защиты. Результаты сравнения алгоритмов представлены в таблице.

Таблица

Результаты сравнения алгоритма метода вектора спада и аддитивного алгоритма

Число средств защиты	Время работы аддитивного алгоритма, мс	Время работы алгоритма вектора спада, мс	Отношение вычисленных значений целевой функции
5	0,0	0,0	1,0
6	0,0	0,0	0,98
7	0,0	0,0	0,93
8	0,0	0,0	0,88
9	0,0	3,2	0,82
10	3,0	0,0	0,84
11	3,2	0,0	0,86
12	6,2	3,2	0,72
13	18,6	0,0	0,84
14	47,0	3,0	0,80
15	78,0	0,0	0,69
16	134,2	3,2	0,74
17	253,2	0,0	0,78
18	390,6	9,4	0,65
19	665,6	3,2	0,72

Число средств защиты	Время работы аддитивного алгоритма, мс	Время работы алгоритма вектора спада, мс	Отношение вычисленных значений целевой функции
20	1543,6	3,2	0,70
21	3112,6	6,2	0,57
22	4806,2	6,2	0,67
23	7846,8	9,4	0,69
24	10981,6	12,2	0,60
25	20929,7	12,6	0,63
26	32202,5	16,0	0,65
27	42089,3	19,0	0,70
28	67450,0	22,0	0,66
29	133987,6	21,8	0,57



Сравнительный анализ скорости работы алгоритмов

Измерения скорости работы алгоритмов приведены на рисунке. Данный график подтверждает экспоненциальную зависимость трудоемкости аддитивного алгоритма и полиномиальную зависимость трудоемкости алгоритма метода вектора спада от числа угроз.

Листинг 1. Текст программы на языке Java

```
import java.util.Arrays;

/**
* Реализация алгоритма метода вектора спада
*/
public class GradientSearch {
/** Исходные данные */
private double[] threatProb;
private double[] threatDamage;
private double[] toolPrice;
private double[][] preventionProb;
private double minDamage;

/** Счетчик числа шагов */
```

```
private int stepCount;
/** Значения переменных, вычисленные в результате работы алгоритма */
private int[] optChoice;
/** Время работы алгоритма */
private long time;

/**
* Инициализация исходными данными
*
* @param threatProb массив возможностей проявления угроз
* @param threatDamage массив ущербов от непредотвращения угроз
* @param toolPrice массив стоимостей средств защиты
* @param preventionProb матрица возможностей предотвращения угроз
* @param minDamage минимальный предотвращенный ущерб
*/
public void init(double[] threatProb,
double[] threatDamage,
double[] toolPrice,
double[][] preventionProb,
double minDamage) {
this.threatProb = threatProb;
this.threatDamage = threatDamage;
this.toolPrice = toolPrice;
this.preventionProb = preventionProb;
this.minDamage = minDamage;
}

/**
* Запуск алгоритма
*/
public void run() {
long start = System.currentTimeMillis();
int[] optChoice = new int[toolPrice.length];

Arrays.fill(optChoice, 1);
step(optChoice, objectiveFunc(optChoice));

time = System.currentTimeMillis() - start;
}

/**
* Один шаг работы алгоритма
*/
private void step(int[] optChoice, double value) {
stepCount++;
int index = -1;
double minValue = value;
// поиск точки окрестности с наибольшим
// значением компоненты вектора спада
for (int i = 0; i < toolPrice.length; i++) {
if (optChoice[i] == 1) {
```

```
optChoice[i] = 0;
if (constraintFunc(optChoice) >= minDamage) {
double newMinValue = value - toolPrice[i];
if (newMinValue < minValue) {
minValue = newMinValue;
index = i;
}
}
optChoice[i] = 1;
}
}
// если точка была найдена, переходим на следующий шаг
if (index != -1) {
optChoice[index] = 0;
step(optChoice, minValue);
}
}

/**
* Получение значений переменных,
* вычисленные в результате работы алгоритма
*/
public int[] getOptChoice() {
return optChoice;
}

/**
* Получение времени работы алгоритма
*/
public long getTime() {
return time;
}

/**
* Получение числа шагов
*/
public int getStepCount() {
return stepCount;
}

/**
* Метод для вычисления целевой функции
*/
public double objectiveFunc(int[] choice) {
double result = 0;
for (int i = 0; i < toolPrice.length; i++) {
result += toolPrice[i] * choice[i];
}
return result;
}
```

```
/**
* Метод для вычисления функции ограничений
*/
public double constraintFunc(int[] choice) {
double result = 0;
for (int threat = 0; threat < threatProb.length; threat++) {
double maxPreventionProb = 0;
for (int tool = 0; tool < toolPrice.length; tool++) {
double tmp = preventionProb[threat][tool] *
choice[tool];
if (tmp > maxPreventionProb) {
maxPreventionProb = tmp;
}
}
result += threatDamage[threat] *
threatProb[threat] *
maxPreventionProb;
}
return result;
}
}
```

Выводы. Рассмотрен метод вектора спада. Данный метод относится к группе градиентных методов и предназначен для решения дискретных оптимизационных задач, он дает локальное решение.

Приведено общее описание метода, а также подготовлена программа на языке Java, реализующая алгоритм метода вектора спада применительно к решению задачи выбора вариантов защиты от угроз безопасности вычислительной сети предприятия.

Скорость работы рассмотренного алгоритма и точность полученных результатов сравнивались с показателями работы точного алгоритма. В качестве точного алгоритма был выбран аддитивный алгоритм Балаша.

Подтверждена экспоненциальная зависимость трудоемкости аддитивного алгоритма и полиномиальная зависимость трудоемкости алгоритма метода вектора спада от числа угроз, чем обусловлена существенно более высокая скорость работы последнего. Применение точного алгоритма неприемлемо при решении задачи с большим числом рассматриваемых вариантов защиты.

Платой за высокую скорость работы является низкая точность решений, найденных алгоритмом вектора спада, что может послужить препятствием для его использования.

СПИСОК ЛИТЕРАТУРЫ

1. Г э р и М., Д ж о н с о н Д. Вычислительные машины и труднорешаемые задачи. – М.: Мир, 1982. – 416 с.

2. Пападимитриу Х., Стайглиц К. Комбинаторная оптимизация. Алгоритмы и сложность. – М.: Мир, 1985. – 512 с.
3. Генс Г. В., Левнер Е. В. Эффективные приближенные алгоритмы для комбинаторных задач. – Препринт. – М.: ЦЭМИ АН СССР, 1981. – 66 с.
4. Кофман А., Анри-Лабордер А. Методы и модели исследования операций. Целочисленное программирование. – М.: Мир, 1977.
5. Сергиенко И. В., Лебедева Т. Т., Рощин В. А. Приближенные методы решения дискретных задач оптимизации. – Киев: Наук. думк., 1980. – 276 с.
6. Ростовцев Ю. Г. Основы построения автоматизированных систем сбора и обработки информации. – МО СССР, 1992. – 640 с.
7. Балаш Э. Аддитивный алгоритм для решения задач линейного программирования с переменными, принимающими значение 0 или 1 // Кибернетический сборник. – 1969. – Вып. 6. – С. 217–252.

Статья поступила в редакцию 12.05.2007

Андрей Игоревич Овчинников родился в 1982 г., окончил в 2005 г. МГТУ им. Н.Э. Баумана. Аспирант кафедры “Информационная безопасность” МГТУ им. Н.Э. Баумана.

A.I. Ovchinnikov (b. 1982) graduated from the Bauman Moscow State Technical University in 2005. Post-graduate of “Information Security” department of the Bauman Moscow State Technical University.

Николай Викторович Медведев родился в 1954 г., окончил в 1977 г. МГТУ им. Н.Э. Баумана. Канд. техн. наук, зав. кафедрой “Информационная безопасность” МГТУ им. Н.Э. Баумана. Автор около 50 научных работ в области исследования и разработки защищенных систем автоматической обработки информации.

N.V. Medvedev (b. 1954) graduated from the Bauman Moscow Higher Technical School in 1977. Ph. D. (Eng.), head of “Information Security” department of the Bauman Moscow State Technical University. Author of about 50 publications in the field of study and development of protected systems of automated data processing.

Александр Юрьевич Быков родился в 1969 г., окончил в 1991 г. ВИКИ им. А.Ф. Можайского. Канд. техн. наук, доцент кафедры “Информационная безопасность” МГТУ им. Н.Э. Баумана. Автор около 20 научных работ в области информационной безопасности и исследования систем обработки информации и управления.

A.Yu. Bykov (b. 1969) graduated from the Military Institute for Engineering and Space n. a. A.F. Mozhaiskii in 1991. Ph. D. (Eng.), assoc. professor of “Information Security” department of the Bauman Moscow State Technical University. Author of about 20 publications in the field of study and development of data security and study of systems of data processing and control.